ARMY RESEARCH LABORATORY

Reading, Writing, and Parsing Text Files Using C++

by Robert J. Yager

ARL-TN-545 **June 2013**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TN-545 **June 2013**

Reading, Writing, and Parsing Text Files Using C++

Robert J. Yager
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
June 2013	Final	January 2012—March 2013

4. TITLE AND SUBTITLE
Reading, Writing, and Parsing Text Files Using C++

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Robert J. Yager

5d. PROJECT NUMBER
AH80

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-WML-A
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-545

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Text files are often used to store tabulated data and user-modifiable inputs for scientific modeling. This report presents a set of functions, written in C++, that can be used to read, write, and parse text files.

15. SUBJECT TERMS
C++, read, write, parse, text files

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Robert J. Yager
Unclassified	Unclassified	Unclassified	UU	18	19b. TELEPHONE NUMBER (*Include area code*) 410-278-6689

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

Acknowledgments

The author would like to thank Dr. Benjamin Breech of the U.S. Army Research Laboratory's Weapons and Materials Research Directorate. Dr. Breech provided technical and editorial recommendations that improved the quality of this report.

1. Introduction

Text files are often used to store tabulated data and user-modifiable inputs for scientific modeling. This report presents a set of functions, written in C++, that can be used to read, write, and parse text files. A summary sheet is provided at the end of this report. It presents the yIo namespace, which contains the six functions that are described in this report.

2. Reading and Writing Text Files

2.1 ReadTextFile() Function

The ReadTextFile() function can be used to create a character array that contains all of the information from a text file.

Note that the ReadTextFile() function uses the "new" command to allocate memory for the character array that is pointed to by the return value. Thus, to avoid memory leaks, each use of the ReadTextFile() function should be accompanied by a use of the "delete[]" operator.

ReadTextFile() Code

```
inline char* ReadTextFile(//<=========READS A TEXT FILE INTO A CHARACTER ARRAY
    const char* filename){//<------------------------THE NAME OF THE TEXT FILE
  FILE* f=fopen(filename,"rb");//...........binary is necessary to get n right
  if(!f)printf("\nCan't open \"%s\".\n",filename),exit(1);//........is f open?
  size_t n;/*<-*/fseek(f,0,SEEK_END),n=ftell(f),rewind(f);//..get size of file
  char* b=new char[n+1];/*<-*/fread(b,1,n,f),fclose(f),b[n]='\0';//.....read f
  return b;//.....................note that b points to newly allocated memory
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

ReadTextFile() Parameters

filename **filename** specifies the name of a text file.

ReadTextFile() Return Value

The ReadTextFile() function returns a pointer to the beginning of a character array that stores all of the information from the input file.

If the file that is specified by **filename** cannot be opened, the ReadTextFile() function calls the exit() function with status code 1. Inability to open a file is typically the result of an incorrectly specified filename or path.

2.2 WriteTextFile() Function

The WriteTextFile() function can be used to write a character array to a text file.

WriteTextFile() Code

```
inline size_t WriteTextFile(//<========WRITES A CHARACTER ARRAY TO A TEXT FILE
    const char* filename,//<-------------------------THE NAME OF THE TEXT FILE
    const char* text,//<--THE CHARACTER ARRAY THAT WILL BE WRITTEN TO THE FILE
    const char* mode="w"){//<-----USE "w" TO OVERWRITE THE FILE, "a" TO APPEND
  FILE* f=fopen(filename,mode);
  if(!f)printf("\nCan't open \"%s\".\n",filename),exit(1);//........is f open?
  size_t n=fwrite(text,1,strlen(text),f);/*&*/fclose(f);//..write file & close
  return n;//.........................number of characters written to the file
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~~
```

WriteTextFile() Parameters

filename — **filename** specifies the name of the text file that will be written.

text — **text** points to a character array that contains the text that will be written to the text file.

mode — **mode** specifies how the text will be written to the output file. Use “w” to overwrite an existing file. Use “a” to append to the end of an existing file. In either case, if a file with the same name as the output filename doesn’t already exist, a new file will be created. The default value is “w.”

WriteTextFile() Return Value

The WriteTextFile() function returns the number of characters that were successfully written to the output file.

If the file that is specified by **filename** cannot be opened, the WriteTextFile() function calls the exit() function with status code 1. Inability to open a file is often the result of an incorrectly specified filename or path. However, it can also be the result of a file being marked as read only, as may be the case if a file is open in another program.

2.3 ReadTextFile()/WriteTextFile() Example

The following example first creates a file named “example.txt,” then reads the newly created file and displays its contents:

```
#include "y_io.h"//.....................................<string>,<cstdio>
int main(){
  yIo::WriteTextFile("example.txt","Hello World!");
  char* s=yIo::ReadTextFile("example.txt");
  printf("%s\n",s);//.....................display the contents of "example.txt"
  delete[] s;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

OUTPUT:

```
Hello World!
```

---

3. Comment Removal

3.1 RemoveLineComments() Function

The RemoveLineComments() function can be used to overwrite line comments in a character array. A line comment is a comment that begins with some identifying set of characters and continues to the end of the line. Memory that is occupied by line comments isn't actually freed by the RemoveLineComments() function. Instead, all of the line-comment characters are replaced with a user-specified character.

RemoveLineComments() Code

```
  inline char* RemoveLineComments(//<==================OVERWRITES LINE COMMENTS
      char* text,//<------------------A CHARACTER ARRAY CONTAINING LINE COMMENTS
      const char* start="#",//<----------------------START-OF-COMMENT INDICATOR
      char c=' '){//<------THE CHARACTER THAT WILL BE USED TO OVERWRITE COMMENTS
    char* s=text;
    while(s=strstr(s,start))memset(s,c,strcspn(s,"\n\f\r"));
    return text;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

RemoveLineComments() Parameters

text	**text** points to a character array that contains the text that will have its line comments overwritten. Typically, the character array is created using the ReadTextFile() function.
start	**start** is used to identify the beginning of a line comment. The default value is "#."
c	**c** specifies the character that will be used to replace the characters in the line comment. The default value is a space.

RemoveLineComments() Return Value

The RemoveLineComments() function returns the input pointer **text**. Although the pointer **text** is unmodified, the character array that **text** points to is modified.

3.2 RemoveBlockComments() Function

The RemoveBlockComments() function can be used to overwrite block comments in a character array. A block comment is a comment that begins with some identifying set of characters and ends with a different set of identifying characters. Block comments may or may not span multiple lines. Note that the memory that is occupied by block comments isn't actually freed by the RemoveBlockComments() function. Instead, all of the block-comment characters are replaced with a user-specified character.

RemoveBlockComments() Code

```
inline char* RemoveBlockComments(//<=================OVERWRITES BLOCK COMMENTS
    char* text,//<-----------------A CHARACTER ARRAY CONTAINING BLOCK COMMENTS
    const char* start="/*",//<----------------------START-OF-COMMENT INDICATOR
    const char* end="*/",//<-------------------------END-OF-COMMENT INDICATOR
    char c=' '){//<------THE CHARACTER THAT WILL BE USED TO OVERWRITE COMMENTS
  char* s=text;
  while(s=strstr(s,start)){
    int t=strstr(s,end)-s+strlen(end);/*<-*/if(t<0)t=0;
    memset(s,c,t);}
  return text;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

RemoveBlockComments() Parameters

text	**text** points to a character array that contains the text that will have its block comments overwritten. Typically, the character array is created using the ReadTextFile() function.
start	**start** indicates the beginning of a block comment. The default value is "/*."
end	**end** indicates the end of a block comment. The default value is "*/."
c	**c** specifies the character that will be used to replace the characters in the line comment. The default value is a space.

RemoveBlockComments() Return Value

The RemoveBlockComments() function returns the input pointer **text**. Although the pointer **text** is unmodified, the character array that **text** points to is modified.

3.3 RemoveLineComments()/RemoveBlockComments() Example

The following example begins by creating a character array from a text file that contains both line comments and block comments. Then, both the RemoveLineComments() and the

RemoveBlockComments() functions are used to remove comments from the character array. The contents of the character array are displayed at each step.

```
#include "y_io.h"//.....................................................<cstdio>
int main(){
  char* s=yIo::ReadTextFile("comment_example.txt");
  printf("ORIGINAL TEXT:\n%s\n\n",s);
  printf("LINE COMMENTS REMOVED:\n%s\n\n",yIo::RemoveLineComments(s));
  printf("LINE COMMENTS AND BLOCK COMMENTS REMOVED:\n%s\n\n",
    yIo::RemoveBlockComments(s));
  delete[] s;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~
```

comment_example.txt:

```
#sample comment
Hello/*sample comment*/ World!
```

OUTPUT:

```
ORIGINAL TEXT:
#sample comment
Hello/*sample comment*/ World!


LINE COMMENTS REMOVED:

Hello/*sample comment*/ World!


LINE COMMENTS AND BLOCK COMMENTS REMOVED:

Hello                   World!
```

---

4. Parsing Character Arrays

4.1 Parse() Function

The Parse() function can be used to separate the text contained in a character array into a set of smaller character arrays called tokens. Tokens are separated by user-defined delimiting characters. Common delimiters are commas, tabs, and spaces.

The Parse() function works by searching for tokens in the character array pointed to by the input variable **text**. When a token is found, the delimiting character that immediately follows the token is replaced by a NULL character. A pointer to the beginning of the token is stored in a vector.

Parse() Code

```
inline vector<char*> Parse(//<=====================================1D PARSER
    char* text,//<---------------------THE CHARACTER ARRAY THAT WILL BE PARSED
    const char* delimiters=" ,\t\n\f\r"){//<---CHARACTERS THAT SEPARATE TOKENS
  vector<char*> S(1,text=strtok(text,delimiters));//....output array of tokens
  while(text=strtok(NULL,delimiters))S.push_back(text);
  return S;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

Parse() Parameters

text — **text** points to the character array that will be parsed. Typically, the character array is created using the ReadTextFile() function.

delimiters — **delimiters** specifies a set of characters that separate tokens. Any set of consecutive delimiter characters will act as a single delimiter. For example, suppose that the text string "8,,,9" was parsed using the Parse() function and with **delimeters** set to ",". The Parse() function will treat the set of three consecutive commas as a single delimiter. Thus, only two tokens will be found. By default, spaces, commas, tabs, line feeds, form feeds, and carriage returns are all treated as delimiters.

Parse() Return Value

The Parse() function returns a vector of pointers. Each pointer points to the beginning of a token. The tokens are stored in the character array that was originally pointed to by the **text** input parameter.

4.2 Parse2D() Function

The Parse2D() function can be used to separate the text contained in a character array into a set of smaller character arrays called tokens. Tokens are separated by two types of user-defined delimiting characters. The first set of delimiters separates tokens within a row of data. Common examples are spaces, commas, and tabs. The second set of delimiters separate data rows. Common examples are line feeds, form feeds, and carriage returns.

The Parse2D() function works by searching for row ends in the character array pointed to by the input variable **text**. When a row end is found, the delimiting character that immediately follows the row is replaced by a NULL character. The Parse() (not 2D) function is then used to parse the row.

Parse2D() Code

```
inline vector<vector<char*> > Parse2D(//<==========2D PARSER (CALLS 1D PARSER)
    char* text,//<---------------------THE CHARACTER ARRAY THAT WILL BE PARSED
    const char* delimiters=" ,\t",//<--------------PRIMARY (COLUMN) DELIMITERS
    const char* ends="\n\f\r"){//<------------------SECONDARY (ROW) DELIMITERS
  vector<vector<char*> >S;//.............................output array of tokens
  char* b,* c=new char[strlen(delimiters)+strlen(ends)+1];
  strcpy(c,delimiters),strcat(c,ends);
  while(*(b=text+strspn(text,c))){//.............................find row start
    text=b+strcspn(b,ends),*text=0,text++;//.......................find row end
    S.push_back(Parse(b,delimiters));}//.............................parse row
  delete[] c;
  return S;//...the number of columns per row may not be the same for all rows
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~
```

Parse2D() Parameters

text	**text** points to the character array that will be parsed. Typically, the character array is created using the ReadTextFile() function.
delimiters	**delimiters** specifies a set of characters that separate tokens within the same row. Any set of consecutive delimiting characters act as a single delimiter. Default values are spaces, commas, and tabs.
ends	**ends** specifies a set of characters that separate rows of tokens. Default values are line feeds, form feeds, and carriage returns.

Parse2D() Return Value

The Parse2D() function returns a vector of vectors of pointers. Each pointer points to the beginning of a token. The tokens are stored in the character array that was originally pointed to by the **text** input parameter.

4.3 Parse()/Parse2D() Example

The following example begins by creating a character array from a text file that contains a table of numbers. The example parses the character array first using the Parse() function, then using the Parse2D function. Between the two parsings, the character array is deleted, then recreated.

```
#include "y_io.h"//.....................................<cstdio>,<vector>
int main(){
  char* s=yIo::ReadTextFile("parse_example.txt");
  printf("ORIGINAL TEXT:\n%s\n\n",s);
  std::vector<char*> A=yIo::Parse(s);
  printf("PARSED TEXT:\n");
  for(int i=0;i<9;++i)printf("%s , ",A[i]);
  delete[] s;
  s=yIo::ReadTextFile("parse_example.txt");
  std::vector<std::vector<char*> > B=yIo::Parse2D(s);
  printf("\b\b \n\n2D-PARSED TEXT:\n");
  for(int i=0;i<3;++i)for(int j=0;j<3;++j)
  printf("%s , %s",B[i][j],j==2?"\b\b \n":"");
  delete[] s;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

parse_sample.txt:

```
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0
```

OUTPUT:

```
ORIGINAL TEXT:
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0

PARSED TEXT:
1.0 , 2.0 , 3.0 , 4.0 , 5.0 , 6.0 , 7.0 , 8.0 , 9.0

2D-PARSED TEXT:
1.0 , 2.0 , 3.0
4.0 , 5.0 , 6.0
7.0 , 8.0 , 9.0
```

5. Summary

A summary sheet is provided at the end of this report. It presents the yIo namespace, which contains the six functions that are described in detail in this report. Also presented is an example that can be used to test the performance of the functions contained in the yIo namespace.

yIo Summary

y_io.h

```
#ifndef Y_IO_H_
#define Y_IO_H_
#include <vector>
#include <cstdio>//..FILE,fopen(),printf(),fseek(),SEEK_END,ftell(),...,fclose()
#include <cstdlib>//.....................................................exit()
#include <cstring>//.......................strstr(),memset(),strlen(),strtok()
namespace yIo{
  using std::vector;
  //@@@@@@@@@@@@@@@@@@@@@@@@ READ AND WRITE TEXT FILES @@@@@@@@@@@@@@@@@@@@@@@@@@@@
  inline char* ReadTextFile(//<=========READS A TEXT FILE INTO A CHARACTER ARRAY
      const char* filename){//<----------------------THE NAME OF THE TEXT FILE
    FILE* f=fopen(filename,"rb");//...........binary is necessary to get n right
    if(!f)printf("\nCan't open \"%s\".\n",filename),exit(1);//........is f open?
    size_t n;/*<-*/fseek(f,0,SEEK_END),n=ftell(f),rewind(f);//..get size of file
    char* b=new char[n+1];/*<-*/fread(b,1,n,f),fclose(f),b[n]='\0';//.....read f
    return b;//.....................note that b points to newly allocated memory
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
  inline size_t WriteTextFile(//<========WRITES A CHARACTER ARRAY TO A TEXT FILE
      const char* filename,//<-------------------------THE NAME OF THE TEXT FILE
      const char* text,//<--THE CHARACTER ARRAY THAT WILL BE WRITTEN TO THE FILE
      const char* mode="w"){//<-----USE "w" TO OVERWRITE THE FILE, "a" TO APPEND
    FILE* f=fopen(filename,mode);
    if(!f)printf("\nCan't open \"%s\".\n",filename),exit(1);//........is f open?
    size_t n=fwrite(text,1,strlen(text),f);/*&*/fclose(f);//..write file & close
    return n;//.........................number of characters written to the file
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
  //@@@@@@@@@@@@@@@@@@@@@@@@@@ COMMENT REMOVAL @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  inline char* RemoveLineComments(//<==================OVERWRITES LINE COMMENTS
      char* text,//<-----------------A CHARACTER ARRAY CONTAINING LINE COMMENTS
      const char* start="#",//<-----------------------START-OF-COMMENT INDICATOR
      char c=' '){//<------THE CHARACTER THAT WILL BE USED TO OVERWRITE COMMENTS
    char* s=text;
    while(s=strstr(s,start))memset(s,c,strcspn(s,"\n\f\r"));
    return text;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
  inline char* RemoveBlockComments(//<=================OVERWRITES BLOCK COMMENTS
      char* text,//<-----------------A CHARACTER ARRAY CONTAINING BLOCK COMMENTS
      const char* start="/*",//<----------------------START-OF-COMMENT INDICATOR
      const char* end="*/",//<------------------------END-OF-COMMENT INDICATOR
      char c=' '){//<------THE CHARACTER THAT WILL BE USED TO OVERWRITE COMMENTS
    char* s=text;
    while(s=strstr(s,start)){
      int t=strstr(s,end)-s+strlen(end);/*<-*/if(t<0)t=0;
      memset(s,c,t);}
    return text;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
  //@@@@@@@@@@@@@@@@@@@@@@@@@@ STRING PARSERS @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  inline vector<char*> Parse(//<=======================================1D PARSER
      char* text,//<---------------------THE CHARACTER ARRAY THAT WILL BE PARSED
      const char* delimiters=" ,\t\n\f\r"){//<---CHARACTERS THAT SEPARATE TOKENS
    vector<char*> S(1,text=strtok(text,delimiters));//....output array of tokens
    while(text=strtok(NULL,delimiters))S.push_back(text);
    return S;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
  inline vector<vector<char*> > Parse2D(//<==========2D PARSER (CALLS 1D PARSER)
      char* text,//<--------------------THE CHARACTER ARRAY THAT WILL BE PARSED
      const char* delimiters=" ,\t",//<--------------PRIMARY (COLUMN) DELIMITERS
      const char* ends="\n\f\r"){//<------------------SECONDARY (ROW) DELIMITERS
    vector<vector<char*> >S;//.........................output array of tokens
    char* b,* c=new char[strlen(delimiters)+strlen(ends)+1];
    strcpy(c,delimiters),strcat(c,ends);
    while(*(b=text+strspn(text,c))){//.............................find row start
      text=b+strcspn(b,ends),*text=0,text++;//.......................find row end
      S.push_back(Parse(b,delimiters));}//.............................parse row
    delete[] c;
    return S;//...the number of columns per row may not be the same for all rows
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
}//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#endif
```

EXAMPLE

```
#include <ctime>
#include <cmath>
#include "y_io.h"//.........................................<vector>,<cstdio>
inline double ElapsedTime(){
  static int clocks=0;
  double time=double(clock()-clocks)/CLOCKS_PER_SEC;
  clocks=clock();
  return time;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
int main(){
  int M=10000,N=100;//.....................number of rows and columns in array
  printf("----------------------TEST SETUP-------------------------\n");
  printf("         number of columns (N) in s1,s2,s3: %d\n",N);
  printf("            number of rows (M) in s1,s2,s3: %d\n",M);
  printf("                  Characters per element: %d\n",10);
  printf("------CREATE s1, FILL WITH RANDOM #S, & WRITE TO FILE-----\n");
  char* s1=new char[M*N*10+1];//.............................storage for array
  for(int i=0;i<M;++i)for(int j=0;j<N;++j,s1+=10)
    sprintf(s1,"%9.5f%s",rand()/double(RAND_MAX)/100,j==N-1?"\n":",");
  *s1=0,s1-=(M*N*10);//..........terminate s, then reset s to beginning of array
  printf("%15stime to create s1=%8.3f seconds\n","",ElapsedTime());
  yIo::WriteTextFile("test.txt",s1);
  printf("  time to write s1 to \"test.txt\"=%8.3f seconds\n",ElapsedTime());
  printf("----CREATE s2, FILL WITH DATA FROM \"test.txt,\" & PARSE----\n");
  char* s2=yIo::ReadTextFile("test.txt");
  printf("%15stime to create s2=%8.3f seconds\n","",ElapsedTime());
  std::vector<std::vector<char*> > B=yIo::Parse2D(s2);
  printf("%16stime to parse s2=%8.3f seconds\n","",ElapsedTime());
  printf("----CREATE s3, FILL WITH PARSED DATA, & COMPARE TO s1-----\n");
  char* s3=new char[M*N*10+1];
  for(int i=0;i<M;++i)for(int j=0;j<N;++j,s3+=10)
    sprintf(s3,"%9s%s",B[i][j],j==N-1?"\n":",");
  *s3=0,s3-=M*N*10;//..........terminate s3, then reset s3 to beginning of array
  printf("%15sTime to create s3=%8.3f seconds\n","",ElapsedTime());
  yIo::WriteTextFile("long_junk2.txt",s3);
  printf("%27ss3=s1?   %s\n","",!strcmp(s1,s3)?"true":"false");
  delete[] s1,delete[] s2,delete[] s3;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~24MAY2013~~~~~~
```

OUTPUT

```
----------------------TEST SETUP-------------------------
         number of columns (N) in s1,s2,s3: 100
            number of rows (M) in s1,s2,s3: 10000
                  Characters per element: 10
------CREATE s1, FILL WITH RANDOM #S, & WRITE TO FILE-----
                time to create s1=   0.670 seconds
  time to write s1 to "test.txt"=   0.047 seconds
----CREATE s2, FILL WITH DATA FROM "test.txt," & PARSE----
                time to create s2=   0.000 seconds
                 time to parse s2=   0.109 seconds
----CREATE s3, FILL WITH PARSED DATA, & COMPARE TO s1-----
                Time to create s3=   0.172 seconds
                           s3=s1?   true
```

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB IMAL HRA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
2 (PDF)	ARDEC RDA REIS AM D ERICSON C DOUROS
1 (PDF)	AMRDEC RDMR SSL T G WIGGS
2 (PDF)	DRTA RDMR SSL T Y SOHN J BELL
1 (PDF)	AFRL/RWW RDMR SSL T C EWING
1 (PDF)	NAWCAD RDMR SSL T E LYNCH
1 (PDF)	NSWCDD RDMR SSL T K LEWIS

ABERDEEN PROVING GROUND

NO. OF COPIES	ORGANIZATION
30 (5 HC, 25 PDF)	DIR USARL
	RDRL SLB W
	P GILLICH
	L HALL
	T MYERS
	C KENNEDY
	N MOHOLKAR
	K RAFAELS
	RDRL WM
	P BAKER
	RDRL WML
	M ZOLTOSKI
	RDRL WML A
	M ARTHUR
	B BREECH
	P BUTLER
	B FLANDERS
	W OBERLE
	C PATTERSON
	R PEARSON
	L STROHM
	A THOMPSON
	R YAGER (1 PDF, 5 HC)
	RDRL WML B
	N TRIVEDI
	RDRL WML C
	S AUBERT
	RDRL WML D
	R BEYER
	RDRL WML E
	P WEINACHT
	RDRL WML F
	D LYON
	RDRL WML G
	J SOUTH
	RDRL WML H
	J NEWILL

INTENTIONALLY LEFT BLANK.